```
readme for fibonacci

describes design and implementation for fibonacci
        To start the assignment, I wrote the full program in C.
        I did this to get a "feel" for solving the problem, and to make sure that my
program handled stuff like proper usage and error output.
        Then, I moved the fibonacci function to a seperate file, and used the GNU
compiler to convert it to assembly. However, at the time I was working on a 64-bit
machine running OS X, and the output file was mostly gibberish. I tried it again on a
32-bit ilab machine (alphabits), which produced more human-readable code, but I
realized that it was obvious it had been compiled by a machine, not handwritten. I
began re-writing the code to try and make sense of it, but it was quite a mess.
        A friend of mine noticed what I was doing, and suggested that it would be
easier to just write the program "from scratch", so I attempted that. At this point, I
realized my method of computing the nth fibonacci number was incredibly inefficient, as
I was using recursion, which is complex and "messy" in assembly.
        I began re-writing the assembly file from scratch, with an iterative approach,
and modelled it from a simple c implementation of the iterative algorithm (allocating
memory on the stack for variables, etc).

detail design, design/implementation challenges
        The assembly file itself is organized into four sub-functions: fib, loop,
overflow, and end. Fib sets up the variables and stack pointer, and moves the given
value from memory into a register. The loop does the computation, and checks for
overflow. Overflow puts -1 into the return location, and end returns the value.

analysis of space and time performance of program

Some simple operations are performed in constant time, and the loop is performed n-1
times. The order of the function is n-1, so big-o is n.
```